

6 SERIES 5 DATABASE HANDLING

The Series 5 uses the relational database management system (DBMS) of EPOC32 which supports SQL (Standard Query Language).

Apart from the removed keywords RECSIZE, COMPRESS and ODBINFO, the Series 3c methods of database programming are completely understood by the Series 5 model and existing code will not have to change. However, it is very strongly recommended that you use INSERT, MODIFY, PUT and CANCEL along with bookmarks and transactions, rather than using APPEND, UPDATE, POS and POSITION.

See also the 'Alphabetic Listing' chapter for some more detailed description of the use of new and changed database commands and the 'Database OPX' section in the 'Using OPXs on the Series 5' chapter.

THE SERIES 5 DATABASE MODEL

As has been stressed previously, it is very strongly recommended that you use this Series 5 specific model on the Series 5, despite the fact that the data file handling methods of the Series 3c may still be used on the Series 5. The reasons for this are as follows:

- the new keywords closely reflect the underlying EPOC32 database model supplied by DBMS. They are therefore more efficient than the Series 3c keywords on the Series 5.
- to emulate the Series 3c behaviour, APPEND has to create an intermediate copy of the record which is erased on completion of the keyword. This ensures the rather strange requirement that the field values of the previous APPEND are used as the initial values for the current APPEND. This can make a database grow far larger than on the Series 3c. You can, however, use COMPACT or SETFLAGS to remove erased records from a database.
- without transactions writing a large number of records to a database is far slower on the Series 5 than on the Series 3c. However, with transactions it is far faster.
- the Series 5 model is superior.

DATABASES, TABLES, VIEWS, FIELDS AND FIELD HANDLES

To describe the new model it is necessary to expand upon the terminology that was used in the previous chapter.

A Series 3c data file corresponds more or less to a single *table* in a DBMS database file. A database can contain one or more tables. A table, like a data file on the Series 3c, contains records which are made up of fields. Unlike the Series 3c, however, the field names as well as the table names are stored in the database.

CREATING DATABASES AND TABLES

With the statement:

```
CREATE "datafile",A,f1%,f2%
```

as described in the previous chapter, the Series 5 creates a database called `datafile` and a table with the default name `Table1` would be added to it. The field names are derived from the `f1%` and `f2%` which are called *field handles*. The type of the field, as always, is defined by these handles.

With the Series 5 it is also possible to use, for example,

```
CREATE "people FIELDS name, number TO phoneBook",A,n$,number$
```

This will create a table called `phoneBook` in the database called `people`, creating the database too if it does not exist. The table will have fields `name` and `number`, whose respective types are specified by the field handles `n$` and `number$`, both strings in this example.

Note that CREATE creates a **table**. An error is raised if the table already exists in the database. DBMS does not allow the database to be open when a table (or an index: see the 'Database OPX' section in the 'Using OPXs on the Series 5' chapter) is created in it, so you should first close the database, i.e. close any tables previously opened in it, before using CREATE.

Logical names

You can have up to 26 views on tables open at a time on the Series 5. Each of these must have a logical name: A to Z (the Series 3c only supported 4 files open at one time).

Fields

On the Series 5, field names may be up to 32 characters long, including any qualifier like `&`.

OPENING DATABASES AND TABLES

With the Series 3c OPEN statement,

```
OPEN "datafile",A,f1%,f2%
```

the Series 5 would open the default table Table1 and provide access to as many fields as there are handles supplied.

On the Series 5, it is also possible to open multiple *views* on a table simultaneously and to specify which fields are to be available in a view, e.g.

```
OPEN "people SELECT name FROM phoneBook",A,n$
```

This view gives you access to just the name field from the phoneBook table.

The string from SELECT onwards in the OPEN statement forms an SQL query which is passed straight on to the underlying EPOC32 DBMS. The SQL command-set is specified in Appendix F.

A more advanced view, ordered by an *index* (described later), would be opened as follows,

```
OPEN "people SELECT name,number FROM phoneBook ORDER BY name ASC,  
number DESC",A,n$,num%
```

This would open a view with name fields in ascending alphabetical order and if any names were the same then the number field would be used to order these records in descending numerical order.

TRANSACTIONS

A set of related records should be committed only on successfully PUTting the last one. Otherwise all new records may be discarded using ROLLBACK. This ensures the atomicity of the whole transaction.

Transactions allow changes to a database to be *committed* in stages. It is necessary to use transactions in database operations to achieve reasonable speeds.

Transactions are a truly fundamental part of the DBMS model, so much so that without the use of transactions you will find that writing to a DBMS database is in fact slower than the equivalent operations in on the Series 3c. With transactions however, the Series 5 database handling is far faster than that of the Series 3c.

A transaction is carried out using the following commands:

- BEGINTRANS begins a transaction on the current database. Once a transaction has been started on a view (or table) then all database keywords will function as usual, but the changes to that view will not be made until COMMITTRANS is used.
- COMMITTRANS commits the transaction of the current view.
- ROLLBACK cancels the current transaction on the current view. Changes made to the database with respect to this particular view since BEGINTRANS was called will be discarded.
- INTRANS finds out whether the current view is in a transaction.

RECORD POSITION

In the DBMS model, as with most modern relational database models, absolute record position does not have much significance.

Bookmarks can be assigned to particular records to provide fast record access **and should be used in preference to POS and POSITION** when opening views using the Series 5 OPEN...SELECT... or CREATE...FIELDS... statements. POS and POSITION can be used safely on tables opened or created using a Series 3c-style OPEN or CREATE statement. However, POS and POSITION should **not** be used in conjunction with bookmarks as bookmarks can cause these keywords, kept mainly for Series 3c compatibility, to become inaccurate. Note that if bookmarks are used in conjunction with POS and POSITION accuracy can be restored by using FIRST or LAST on the current view.

The new commands provided for the use of bookmarks are as follows: BOOKMARK puts a bookmark at the current record of the current database view. The value returned can be passed to GOTOMARK to make the record current again and to KILLMARK to delete the bookmark.

SAVING RECORDS

When using the Series 5 extensions to CREATE and OPEN, you should also use the new MODIFY, INSERT, PUT and CANCEL keywords in preference to the APPEND and UPDATE Series 3c commands. APPEND and UPDATE will still work as expected, but do not naturally fit in the DBMS model.

- MODIFY allows records to be changed without being moved to the end of the set (as UPDATE still does).
- Instead of copying the current record to the end of the set as APPEND does, INSERT appends a new record to the end of the set with numeric fields set to 0 and string fields empty if values have not been assigned to them.
- PUT marks the end of a database's INSERT or MODIFY phase and makes the changes permanent.
- CANCEL marks the end of a database's INSERT or MODIFY phase and discards the changes made during that phase.

The number of records

The COUNT function returns the number of records in the file. If you try to count the number of records between assignment and APPEND/UPDATE or between MODIFY/INSERT and PUT an 'Incompatible update mode' error will be raised.

CLOSING VIEWS AND DATABASES

CLOSE closes the current view on a database. If there are no other views open on the database then the database itself will be closed.

INDEXES

Indexes can be constructed on a table using several fields as *keys*. These indexes are subsequently used to provide major speed improvements when opening a table or views on them.

Further database functionality is provided in the Database OPX, discussed in the 'Using OPXs' chapter.

COMPACTION

COMPACT replaces the COMPRESS command on the Series 5. This compacts a database, rewriting the file in place without any removed or deleted data. All views on the database and the hence the file itself should be closed before calling this command. Compaction may also be done automatically on closing a file by setting the automatic compaction flag using SETFLAGS. See the 'Alphabetic Listing' chapter for full details of this.

OPENING A DATABASE CREATED BY THE DATA APPLICATION

It is currently not possible to open an OPL database from the Data application. You can however open a file created by the Data application in an OPL program. The file is opened for reading only because if it were written to, OPL would have to discard all the formatting characters and prevent the Data application from reopening the file subsequently. An OPL program can create a new OPL database and copy the Data application records into it if necessary.

To open a Data application database that has one string field which you need to access, you could use:

```
OPEN "file",a,a$
```

Types not supported by OPL will be ignored. Note that integer fields in the Data application correspond to long integer fields in OPL: the Data application does not support (16-bit) integer fields. The types and order of the OPL field handles must match the fields in the Data file. For example, if the data file Data2 contains:

1. long integer field
2. date/time field (ignored by OPL)
3. string field
4. floating-point number field

you could access the fields supported by OPL using:

```
OPEN "Data2",A, f1&,f2$,f3
```

It would be better, however, to use the SQL SELECT clause to name the required Data file fields explicitly. For this to be possible it is necessary to use table name and the same field names as are used by Data. All Data files have a single table called Table1. The fields (referred to internally as columns in Data) are named ColA1, ColA2, etc.

So, with the field types from the previous example, the Data file could be opened using:

```
OPEN "Data2 SELECT ColA1,ColA3,ColA4 FROM Table1",a,f1&,f2$,f3
```